

A Data Distribution Service in a Hierarchical SDN Architecture: Implementation and Evaluation

Alejandro Llorens-Carrodegua
Department of Network Engineering
Universitat Politècnica de Catalunya
Barcelona, Spain
alejandro.llorens@entel.upc.edu

Cristina Cervelló-Pastor
Department of Network Engineering
Universitat Politècnica de Catalunya
Barcelona, Spain
cristina@entel.upc.edu

Irian Leyva-Pupo
Department of Network Engineering
Universitat Politècnica de Catalunya
Barcelona, Spain
irian.leyva@entel.upc.edu

Abstract—Software-defined networks (SDNs) have caused a paradigm shift in communication networks as they enable network programmability using either centralized or distributed controllers. With the development of the industry and society, new verticals have emerged, such as Industry 4.0, cooperative sensing and augmented reality. These verticals require network robustness and availability, which forces the use of distributed domains to improve network scalability and resilience. To this aim, this paper proposes a new solution to distribute SDN domains by using Data Distribution Services (DDS). The DDS allows the exchange of network information, synchronization among controllers and auto-discovery. Moreover, it increases the control plane robustness, an important characteristic in 5G networks (e.g., if a controller fails, its resources and devices can be managed by other controllers in a short amount of time as they already know this information). To verify the effectiveness of the DDS, we design a testbed by integrating the DDS in SDN controllers and deploying these controllers in different regions of Spain. The communication among the controllers was evaluated in terms of latency and overhead.

Index Terms—software-defined networks, data distribution service, testbed, hierarchical architecture

I. INTRODUCTION

As part of the significant advances that have recently been made in many telematic areas—such as Big Data, edge computing and the Internet of Things (IoT)—along with the large growth in mobile data traffic that is expected according to the latest Cisco forecast [1], we consider that a revolutionary approach in the networking field is necessary to overcome this situation. The necessity of a simple network in which many technologies can coexist has driven network operators and academic institutions to search for a solution to fulfill the strict requirements that suppose a convergent and open network such as 5G.

In this sense, the new architecture must be flexible and programmable to be able to adapt itself to a variety of traffic conditions. Additionally, it needs to be dynamic, scalable, available and capable of processing a large amount of data in real time.

In recent years, Software-Defined Networks (SDNs) have emerged as a new networking paradigm capable of delivering

new methods and means to instantiate network functions and services, reduce expenses and boost performance. These advantages are possible by separating the control and data planes. The control plane is formed by the controller, which manages all the forwarding devices in a unified way. The data plane includes switches and routers that perform their forwarding functions by following a set of rules installed by the controller.

The SDN controllers can be classified into two main groups within an architectural point of view. The first one uses a single controller, for the sake of simplicity, but it represents a bottleneck in the network when the number and size of OpenFlow messages increase. Meanwhile, the second group is focused on distributing controllers to a large-scale network in order to improve the network's scalability and avoid a single point of failure. Considering these groups, the best strategy is a combination of them, namely, to deploy a hierarchical SDN controller architecture.

This type of architecture guarantees scalability as specific functions can be defined in the controllers in order to establish their role within the overall network. In this way, only the traffic that is destined for other networks is sent to the controllers at the top level. In a hierarchical architecture, each controller manages its domain and distributes the necessary data to other controllers. In order to guarantee a good performance in the hierarchy, it is necessary to establish proper communication among controllers.

To the best of our knowledge, there is no standard for SDN controller communication. However, there are several works that focus on formal concepts and theoretical architecture. In this paper, we aim to contribute to the design of the hierarchical SDN control plane by developing a testbed formed by SDN controllers that are federated by means of a Data Distribution Service (DDS). Our implementation represents the first step in guaranteeing a robust control plane for 5G networks as their control elements can distribute network information and respond quickly to network and controller failures. Thus, the main contributions of this paper are as follows:

- A communication mechanism to logically distribute SDN controllers (i.e, flat or hierarchical architecture).

- Auto-discovery of SDN controllers and detection of failure in SDN controllers in a short time frame.
- An authentication procedure to avoid unauthorized publishers introducing incorrect messages.

The remainder of this paper is organized in the following way. Section II describes some related works. Section III presents a brief explication of DDS. The implementation of a DDS application in SDN controllers and the main modules of this application are described in Section IV. The tested proposal is presented in Section V. Section VI introduces a thorough analysis of the simulation results. Finally, Section VII reveals the main conclusions of this paper and states some areas for further investigation.

II. RELATED WORK

According to the revised literature, SDN controllers architecture can be classified into centralized or distributed. Centralized controllers, such as NOX-MT [2], Maestro [3], Beacon [4], Ryu [5] and Floodlight [6], have been designed to achieve a determined throughput necessary for enterprise networks and data centers. However, they represent a bottleneck in the network and may present scaling limitations when the amount of traffic and the number of OpenFlow requests increase.

Several works propose a distributed architecture of SDN controllers to improve network resilience and scalability.

In the paper by Koponen et al. [7], the authors propose a distributed system running on a cluster of one or more physical servers, which may run multiple Onix instances and use Apache ZooKeeper [8] to store and synchronize a data structure called network information base (NIB). As a result, Onix provides scalability and resilience by replicating and distributing NIBs between multiple running instances.

Similarly, Tootoonchian et al. [9] propose HyperFlow as a distributed, event-based control plane for OpenFlow that allows the deployment of any number of controllers in network operators. Hyperflow uses the publish/subscribe paradigm to communicate with the controllers using WheelFS [10] as the distributed file system. Thus, controllers can build a global network view and, simultaneously, take charge of the network.

In [11], the authors propose a communication interface for distribute control plane (CIDC) that allows synchronization and the exchange of notifications as well as services between multiple distributed SDN controllers. This interface is composed of four modules: a *Consumer*, *Producer*, *DataUpdater* and *DataCollector*. In its design, each controller plays the role of a *Consumer* for external events and a *Producer* for local events.

These approaches impose a consistent, network-wide view on the controllers and generate large control traffic, which can affect the network latency and throughput despite their ability to distribute SDN control planes. Other authors propose a hierarchical controller architecture to improve those inconveniences.

In [12], the authors present a two-level architecture of SDN controllers. The bottom layer is formed by the area controllers, which are connected to physical switches and routers. In

the upper layer, there are domain controllers, which control the area controllers as devices and synchronize the global abstracted network view through a distributed database.

Phemius et al. [13], [14] implement an extensible Distributed SDN Control plane (DISCO) on top of the Floodlight controller to establish communication with other controllers. By means of its two key elements—*Messenger* and *Agents*—DISCO performs its functions. The former discovers neighboring controllers and maintains a distributed publish/subscribe communication channel; the latter utilizes this channel to exchange network information among controllers.

In addition to the previous contributions, there are some native solutions in the controllers to communicate SDN control planes. The Open Network Operating System (ONOS) [15] controller follows in the footsteps of previous closed distributed SDN controllers such as Onix. It runs on multiple servers in which each one acts as the master controller for a subset of switches. When the number of switches increases, additional instances can be added to the ONOS cluster to distribute the control plane workload. The network view data model of ONOS is implemented using the Titan [16] graph database and the Cassandra [17] key-value store for the distribution and persistence of the network information. In this way, ONOS can be implemented for network operators as it supports hybrid networks and high speeds in large-scale networks. Similarly, OpenDaylight controllers [18] have integrated the OpenDaylight SDN Interface Application (ODL-SDNi App) [19], [20] into their architectures to distribute SDN controllers. The most important element in this application is the SDNi Wrapper, which is responsible for sharing to and collecting information from the federated controllers. The main limitation of this solution is present in this component because its communication is based on the Border Gateway Protocol (BGP); thus, the communication does not occur in real time, which mainly affects the latency and time recovery. Therefore, the SDNi App is still being improved, and there are some aspects to be tested such as peer-to-peer communication and Quality of service (QoS) information exchange.

It is well-known that distributed controllers for multiple domain architectures entail many challenges such as scalability, performance and fine-grained sharing. Our motivation in this paper is to propose a solution to overcome these problems. Our DDS-based communication mechanism combines the fast and predictable distribution of time-critical data in real time with a configurable mode to announce the type of information to be exchanged in order to achieve fine-grained sharing. Furthermore, the hierarchy of SDN controllers aims to improve the network performance as it defines specific functions for each type of controller.

III. DATA DISTRIBUTION SERVICE: OVERVIEW

A DDS is a middleware protocol and application programming interface (API) standard for data-centric connectivity from the Object Management Group (OMG). This standard addresses the publish/subscribe communication for real-time and embedded systems [21].

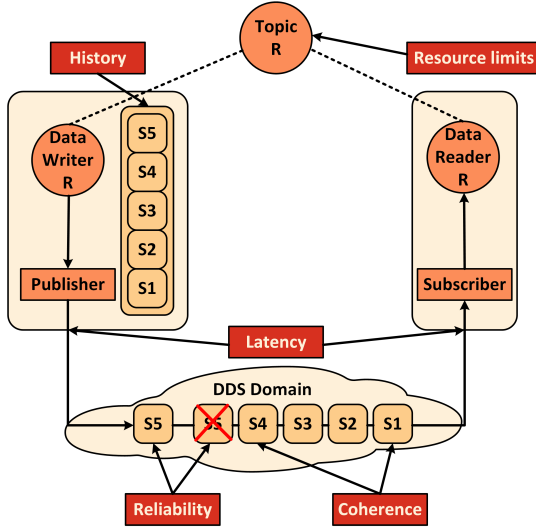


Fig. 1: An overview of the DDS [22].

The DDS provides a global data store in which publishers and subscribers respectively write and read data. In addition, it offers a fast and predictable distribution of time-critical data over a variety of transport networks. Furthermore, the DDS provides a flexible data distribution infrastructure by integrating several types of data sources. Therefore, it delivers a large amount of data with microsecond performance and granular QoS control using a distributed cache, referred to as a data-space. This middleware offers a flexible and modular structure by decoupling location, redundancy, time, message flow and platform.

In Fig. 1, the relation between the following DDS entities is depicted: domain, domain participants, topics, publishers, data writers, subscribers and data readers. The DDS domain represents the global data store containing the information provided by the applications registered to that domain.

A DDS topic describes the type and structure of the data. It represents data streams of the same data type that allow strongly typed data dissemination. The data readers and writers can respectively subscribe and publish specific topics. One or more data readers and writers are respectively managed by subscribers and publishers. Publishers and subscribers discover each other automatically and match if they have compatible topics and QoS.

The topics are exchanged between peers within the DDS domain according to a contract established in the discovery phase. During this phase, each domain participant maintains a local database with all the active data writers and data readers in the same domain. The discovery mechanism is dynamic, so that, the DDS applications do not have to know or configure the endpoints for communications because they are automatically discovered by the DDS.

The OMG DDS standard offers several advantages to users, such as easy integration, performance efficiency, scalability, advanced security, an open standard, an enabled QoS, scalable discovery and applicability.

IV. IMPLEMENTATION OF THE DDS APPLICATION

In this section, we explain the main modules of the DDS Application (DDS App) as well as its behavior in order to offer a more thorough understanding of its performance.

The exchange of network information, the detection of new and failure controllers, and the synchronization among controllers are the main functions of the DDS App. This application is composed of three modules: the publisher, subscriber and synchronization. The modularity of this application offers controllers the ability to simultaneously send and receive network information without any interference or loss of information because this application uses different ports to send and receive information. In the next subsections, we explain how these modules work to guarantee the previous functions.

A. Publisher Module

This module controls the process of sending network information to other controllers. To perform this simple function, the proper configuration of the QoS parameters (i.e. the persistence service, publisher's queue and reliability) is necessary. In this way, we avoid outperforming data consumers and blocking the send queue in the case of slower subscribers. After completing the initial configuration, the publisher needs to check if the connection with the other controllers has been established before it begins to send data.

When the controllers are synchronized, the publisher sends data when there is new data to be sent. The publisher creates the topology topic based on the network information read from the controller data store (e.g. the MD-SAL data store in OpenDaylight controllers). This topic is a data stream composed of eight fields: the *Identifier*, *NodeId*, *Termination-PointId*, *LinkId*, *SourceNode*, *SourceNodeTp*, *DestinationNode* and *DestinationNodeTp*. Each one represents a string of data. Using the identifier field, the publisher announces if it will send a node, a flow or a link.

Finally, the publisher sends the topic to each controller connected to it. This connection is managed by the synchronization module which is explained in Subsection IV-C. The publisher module algorithm is displayed in Fig. 2 to clarify its performance in the controller.

B. Subscriber Module

At its most basic, this module receives the network information sent by other controllers. Similar to the publisher module, it is necessary to configure some QoS parameters to guarantee a well-synchronized connection with the other publisher controllers. However, their initial configurations are different because the subscriber module utilizes listeners to perform its functions.

Some statuses have been configured in this module to recognize when new data is available to be read. Thus, the listener is invoked when these statuses change, avoiding polling the publisher and saving time and resources, which is extremely important to fulfill the latency requirement of 5G networks.

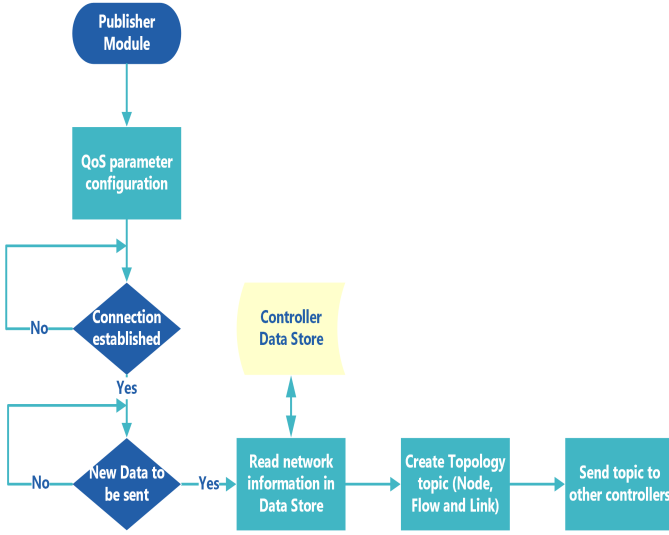


Fig. 2: Publisher Module Algorithm

Once the listener is invoked, the subscriber reads the network information from the topology topic and saves this information in the controller data store, taking into account the identifier field that it has received. The behavior of this module is summarized in its algorithm (see Fig. 3).

C. Synchronization Module

This module is responsible for guaranteeing the synchronization among controllers at all times by considering the controller role in the network. In our previous work, we proposed an SDN controller hierarchy formed by a group of global controllers (GCs) in the upper layer while the bottom layer was composed of area controllers (ACs) [23]. This hierarchy is explained in Section V.

From this starting point, the QoS parameter configuration and the listener initialization must consider the controller

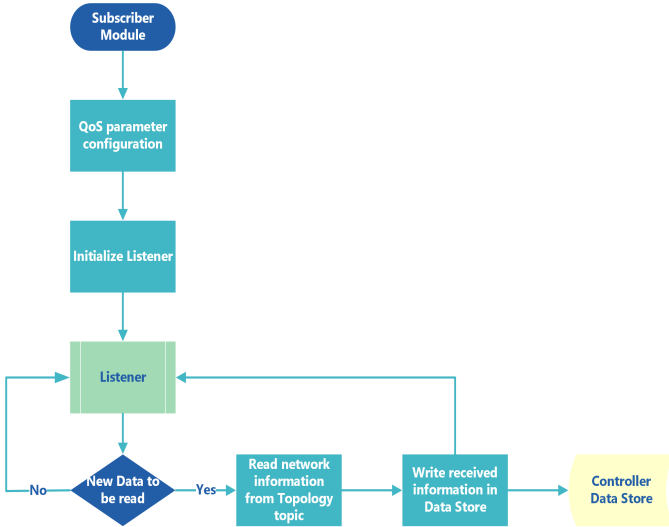


Fig. 3: Subscriber Module Algorithm

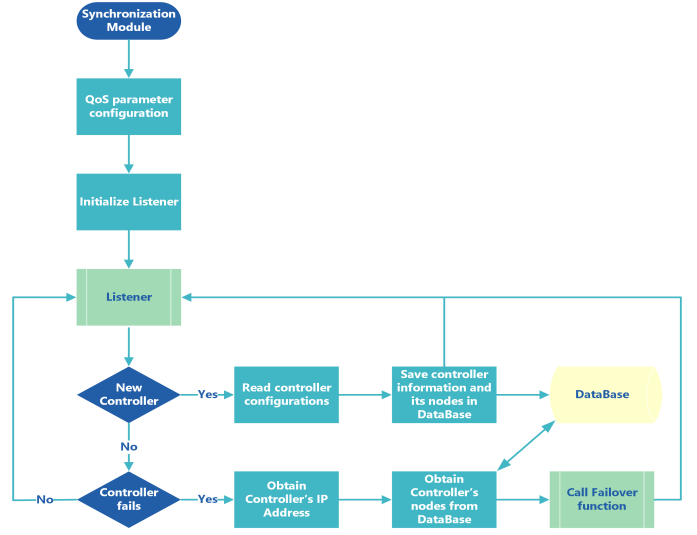


Fig. 4: Synchronization Module Algorithm

role. For this reason, we need to guarantee that the GCs communicate with each other using the DDS to keep a global network view and recognize when their ACs change their statuses to alive or failure. Meanwhile, the ACs can only send or receive network information to or from their GCs. In this sense, our authentication procedure plays an important role during the discovery phase as it ignores any participant that is not compatible with the configured security profiles. Thus, the controllers cannot receive information from entities that have not been registered during the discovery phase.

Once the listener is initialized in the GC, it can detect either a new AC or a failure in one of its existing ACs. In the first case, the GC receives the AC information and saves it to an internal database. In the second case, when it detects that an AC fails, it searches the AC's IP address to correlate it with its assigned nodes. Finally, the module calls a failover function to reassign the nodes to other ACs. The algorithm of this module is depicted in Fig. 4.

V. TESTBED AND EXPERIMENT DESCRIPTION

In this section, we explain the network architecture used during the testbed implementation and describe the experiment to be carried out.

A. SDN Controller Hierarchy

The SDN controller hierarchy is illustrated in Fig. 5. In the upper layer of the hierarchy, there is a group of GCs that are federated using the DDS. They manage and control the ACs, perform load balancing and keep a global network view. Meanwhile, the bottom layer is composed of ACs, which are responsible for the User Plane Functions (UPFs) of the 5G architecture control and flows management.

Both types of controllers use the DDS to exchange network information. The GCs communicate with each other to keep a consistent network state and establish inter-domain flow routes. In the same way, the ACs update their GCs when a

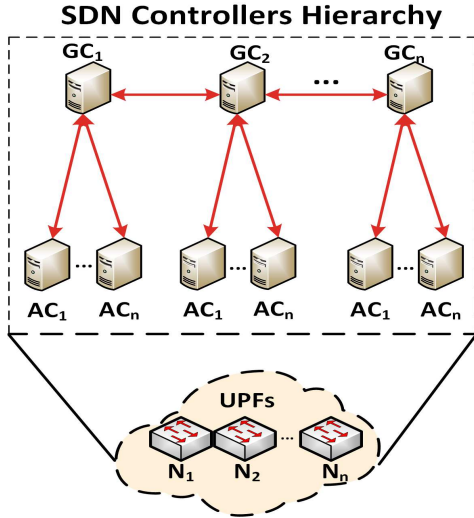


Fig. 5: Hierarchical architecture of SDN Controllers.

change in their topology occurs. Similarly, the GCs inform their ACs when there is a change in the global topology that can affect the communication among the nodes under the control of different ACs.

Furthermore, the use of the DDS allows a stronger performance during the recovery stages because the GCs share their network information with each other. Thus, if any problem arises with a GC operation, its functions are assumed by another GC.

B. Testbed Implementation

Considering the previous SDN controller hierarchy, we built a national SDN testbed formed by OpenDaylight controllers in which the DDS App runs. The OpenDaylight distribution was selected by comparing several features such as the cross-platform compatibility, southbound and northbound interfaces, OpenFlow support, network programmability, efficiency and partnership. In addition, this controller supports a modular framework, providing support for other SDN standards and forthcoming protocols. Furthermore, OpenDaylight applications can collect network information, perform analyses by running algorithms and create new rules throughout the network.

The testbed architecture is composed of two GCs, each of which manages two ACs. The GCs are physically distributed in Granada (University of Granada, UGR) and Barcelona (Universitat Politècnica de Catalunya, UPC). Similarly, their ACs are placed in these locations, and they can only communicate with their GCs. Thus, we have two SDN domains, as is depicted in Fig. 6. The blue dashed line represents the DDS connections over RedIRIS [24].

The GCs were configured to support communication over Wide Area Network (WAN). Some extra steps were necessary to guarantee this capability. First, we set a public IP address for each GC that is announced to other GCs during the discovery stage to establish the connection. Second, we configured the

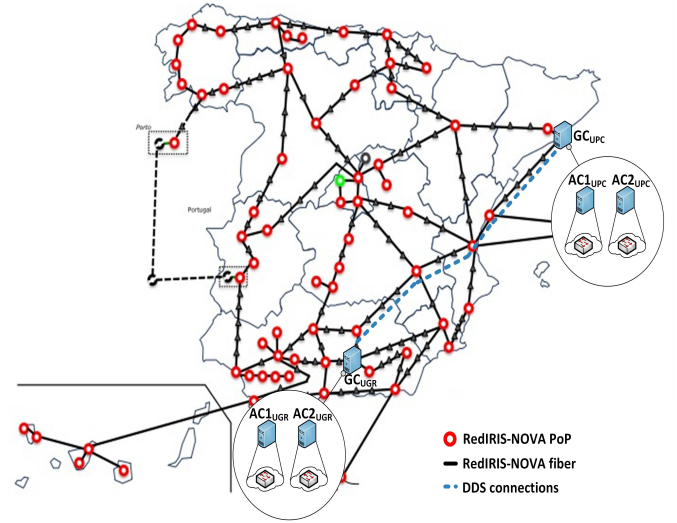


Fig. 6: SDN federated testbed proposal using DDS to communicate with controllers

IP Network Address Translation (NAT) router to allow User Datagram Protocol (UDP) traffic and to map the public IP address to the private Local Area Network (LAN) IP address where the controllers were running (SDN domain). By using port forwarding, we mapped the private ports used to receive discovery and user data traffic to the corresponding public ports. In this way, the GCs can exchange network information and discover other GCs.

The ACs were also configured to support communication over WAN in case their GCs are geographically distant—in another SDN domain. However, the first principle was to use private LAN to communicate with controllers in the same SDN domain.

C. Hardware and Software Configuration

The SDN controllers have been instantiated in virtual machines (VMs) with 1 CPU, 2GB of RAM and 20GB of hard disk. The UPC SDN domain runs over a Mitaka OpenStack-based cloud that is formed by six servers providing the following pool of resources: 48 VCPUs, 256GB of RAM and 8TB of disk storage; all servers have two Gigabit Ethernet (GbE) interfaces and are connected through GbE switches. Meanwhile, the UGR SDN domain is hosted by an OpenStack Mitaka environment, which is executed by a server with the following resources: Intel core i7-6700K CPU@4GHz with 4 VCPUs, 32GB of RAM and a 10Gbps network Ethernet interface.

We used the OpenDaylight controller in this implementation and the Open vSwitch 2.3.0 with support for OpenFlow 1.2, 1.3 and 1.4 as the software switch. The Mininet network emulator was used to create the SDN topologies, and the RTI Connex DDS 5.2.3 was installed in the same VMs where the controllers were running. The classes and methods of the *nddsjava.jar* library were utilized to implement an application capable of integrating the DDS software in an OpenDaylight

controller. To build this application, JDK 1.8, Eclipse Luna, and Maven 3.3.3 were used.

D. Experiment Description

We emulated three different networks to evaluate our implementation. Two of them were taken from the Internet Topology Zoo [25] (i.e. the Abilene network and the BtNorthAmerica network). The other one was designed to represent a minimal network configuration formed by two nodes. For each scenario, we used two GCs and two ACs, specifically, GC_{UPC}, GC_{UGR}, AC1_{UPC} and AC2_{UPC}.

To gather statistically meaningful results, we ran 10 independent experiments per scenario. For each run, we started all controller instances and let them discover each other by means of the DDS. Later, we initialized the network topology where the nodes were assigned randomly to both AC1_{UPC} and AC2_{UPC}. After all the nodes connected to their ACs, we injected new traffic flows into the network. Initially, the incoming packets to the switches did not match with any flow in their flow tables as these were empty. Therefore, they sent a *packet_in* message to their ACs so that the controller could install a new flow entry by means of a *flow_mod* message. Hence, the ACs reactively installed forwarding rules in the network. All of these events must be exchanged among controllers in order to guarantee a robust control plane. Once, the AC1_{UPC} and AC2_{UPC} have discovered their assigned nodes, they proceed to install new flows in their nodes and update their topology view. After that, they share this information with the GC_{UPC}. Similarly, the GC_{UPC} processes that information and shares it with its counterpart in the other SDN domain, the GC_{UGR}.

To assess our implementation, we analyzed two metrics that can affect the performance of the controllers in distributed architectures: delay and overhead. The former represents the time from when a controller generates an event until other controllers are aware of the same event. The latter describes the aggregated data rate employed during the exchange of network information among controllers. The delay among controllers is a key parameter in distributed systems. It is affected by the amount of information to be exchanged, the available bandwidth, the propagation delay and the processing time of the controllers.

The propagation delay is assumed to be fixed because it depends on the distance between the devices and the controller locations does not change. Moreover, the processing time depends on the nodes load and processing capacity, so it can be reduced by either performing load balance or upgrading the hardware platform. The available bandwidth has not been limited, so it gives an approximated upper bound on the overhead. In this regard, the controller overhead can be reduced by limiting the available bandwidth.

VI. EVALUATION

In this section, the evaluation results for the three previous scenarios are presented. We evaluate the behavior of the DDS

App in the controllers by taking into account aspects such as recovery time, delay and controller overhead.

In our tests, different events can take place such as the discovery of new nodes and links, and the installation of new flows. Thus, network state changes must be synchronized among controllers. The higher the number of consecutive events, the higher the controller overhead.

The first scenario is called Minimal network and is composed of two nodes which have been assigned to different ACs. The aim of this scenario is to evaluate the synchronization among controllers in small SDN domains. Thus, we evaluate the communication between GCs and emulate a failure in GC_{UGR} by turning it off. Fig. 7 shows the obtained results.

From Fig. 7 can be observed that the delays are maximum (i.e., 80-140 ms) at the beginning of the communication due to the discovery process among the controllers. Namely, the GC_{UPC} has to process the discovery information received from its assigned ACs and synchronize with the GC_{UGR}. These delays decrease after the discovery phase and remain below 50 ms during the rest of the communication. Moreover, the highest values of overhead are found between the GCs (i.e., approximately 600 KB/s), as shown Fig. 7a, due to the overall network information is exchanged. By contrast, the communication between ACs and the GC_{UPC} shows much lower values, around 200-300 KB/s, see Fig. 7b and Fig. 7c.

In Fig. 7a, the moment the GC_{UGR} fails can be noted since there is an interruption in the communication. This connection is recovered after 100 s because it is the time the GC_{UGR} needs to initialize. This time can be reduced by enhancing the hardware capabilities of the server that host the controller. After the communication between both GCs have been reestablished, the GC_{UPC} shares its network information with GC_{UGR} in order to maintain a consistent control plane. In this case, the synchronization delay does not experiment the initial peak as the GC_{UPC} already knows the information related to its ACs and only needs to update the other GC.

To further evaluate the inter-controller communication, a failure in one of the ACs is analyzed. The AC2_{UPC} was turned off at the end of the test. Notice, the values of overhead and delay between the AC2_{UPC} and the GC_{UPC} are zero after 350 s in Fig. 7c. Moreover, this event is reflected in the communication between GCs which has a spike near 950 KB/s in the overhead. This effect is also evidenced in the communication GC_{UPC}-AC1_{UPC} where there is a small fall in the transmission rate. The latter occurs due to the node reassignment process. It is important to mention that our failover mechanism is still under development which is why an evaluation has not been included in this paper.

The second and third scenarios use Abilene and Bt-NorthAmerica as network topologies with 11 and 36 nodes, respectively. Hence, there will be more events due to the ACs manage a greater number of switches. Additionally, they need to discover and install more links and flows than the first scenario. In these cases, we seek to evaluate the effects of distances and network size in the synchronization delay and overhead between the GCs. Fig. 8 shows the obtained results

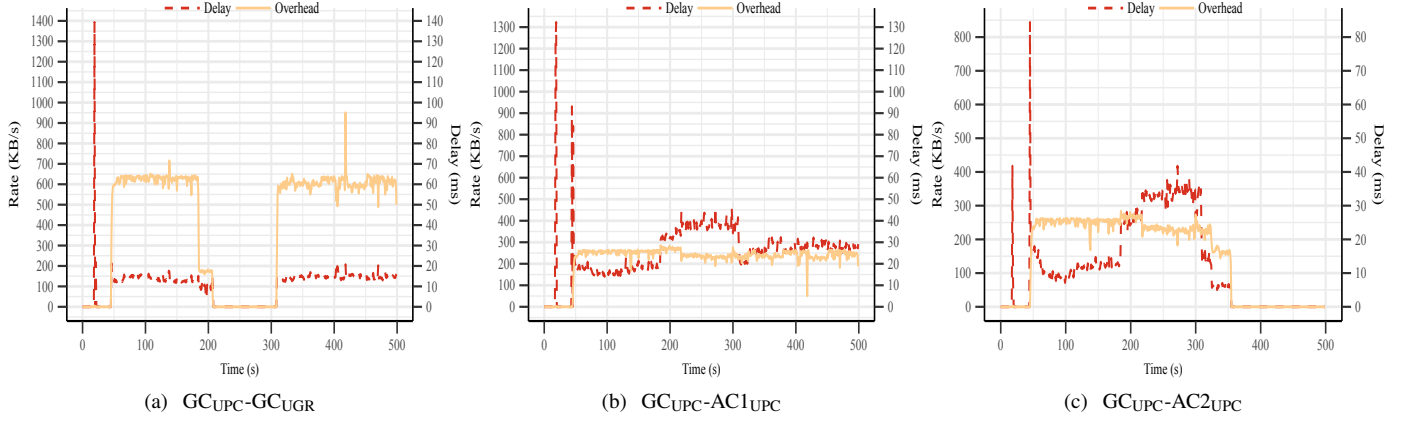


Fig. 7: Synchronization delay and overhead in Minimal network.

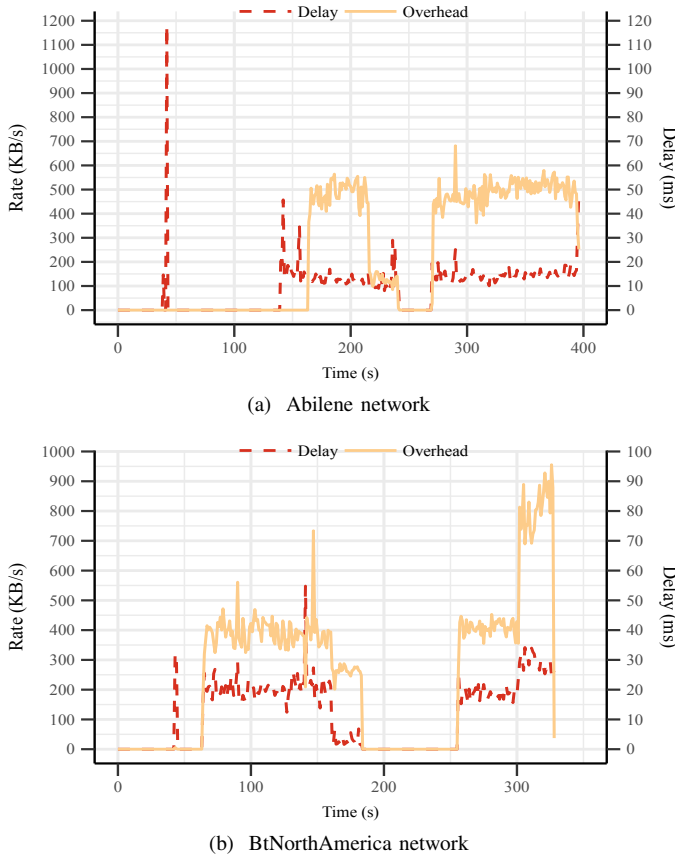


Fig. 8: Synchronization delay and overhead in GC_{UPC} - GC_{UGR} communication.

during these experiments.

Similar to the first scenario, the higher values of delay are obtained during the discovery stage. Once the controllers have synchronized, the delay remains with values around 20 ms. We emulate a fail in the GC_{UGR} in both scenarios as shown by the fall in the transmission rate in Fig. 8. After the recovery, the GC_{UPC} updates the GC_{UGR} on the most recent network

information. In addition, a failure in one of the ACs in the BtNorthAmerica scenario is considered, which is evidenced by a rise in the overhead with values up to 950 KB/s, see Fig. 8b. The obtained results show that the network size does not have a significant impact on the synchronization overhead and delay between the GCs. However, the effects of the AC failure have a longer duration due to the GC_{UPC} needs to reassign a higher number of nodes.

In order to gather the aggregated results for the three evaluated scenarios in terms of the analyzed metrics, a boxplot representation, see Fig. 9, was used which includes a 95% confidence interval based on Student-t distribution. In Fig. 9a, the outliers represent the values of latency related to the discovery phases among controllers. We can see how 75 percentile of the data samples related to the values of latency among the GCs requires less than 15 ms for Minimal and Abilene networks and 25 ms for the BtNorthAmerica scenario. In this sense, we obtain higher values of latency for the third scenario as it generates a large number of events due to the greater number of nodes. Furthermore, we can appreciate how the processing time of the ACs affects their synchronization delay with the GC_{UPC} because they also must configure and manage their assigned nodes. In general, 75 percentile of the data samples require less than 30 ms to synchronize all the events.

From Fig. 9b can be observed that the GC_{UPC} has the highest synchronization overhead among all the controllers. This is expected because it receives all the network information directly from the ACs. Additionally, we can see that 75 percentile of the data samples of both ACs require less than 500 KB/s to share the events with the GC_{UPC} .

Although the obtained results in terms of delay are quite promising, we need to tune some parameters in the DDS App to be in line with the 5G specifications. In this vein, the use of batching techniques in our implementation could decrease the amount of communication overhead associated with the transmission. Thus, our application could collect multiple user data samples to be sent in a single network packet. Hence, we

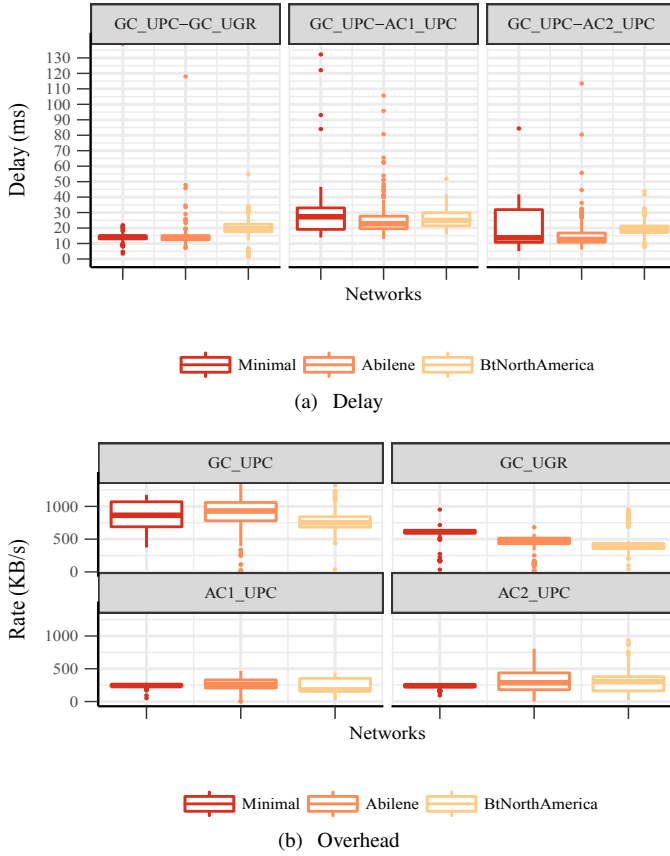


Fig. 9: Aggregated results of the 10 runs per evaluated scenario.

will take advantage of the efficiency of sending larger packets and, at the same time, increase the effective throughput.

VII. CONCLUSION

In this paper, we propose the use of the DDS as a new way to exchange network information among SDN controllers. This approach allows the distribution of time-critical data with a configurable mode to announce the type of information in function of the generated events in the network in order to achieve fine-grained sharing. In addition, our implementation offers the capability of auto-discover SDN controllers and a reliable mechanism to update the network information in case of failure in the controllers. To evaluate the performance of the DDS as a feasible technology for 5G networks where low latency requirements are critical, we have developed a national SDN testbed based on a hierarchical architecture connecting two SDN domains. This architecture is analyzed in three different scenarios by taking into account two metrics: synchronization overhead and delay. The results indicate that the values of latency are very low in most of the cases. We can remark that network information such as nodes, flows and links can be synchronized quickly and lightly by means of the DDS. To reduce the controllers overhead, it is necessary to tune some parameters of the DDS App as a trade-off with the delay is not an option in 5G networks.

In the future, we plan to extend our SDN testbed by connecting more SDN domains and evaluate the DDS App in scenarios where the amount of information to be exchanged is much higher. Moreover, we intend to incorporate Machine Learning techniques to the SDN controllers to optimize the routing of intra-domain and inter-domain traffic flows as well as the node reassignment when controllers fail.

ACKNOWLEDGMENT

The authors would like to thank Prof. Jorge Navarro-Ortiz of University of Granada for cooperating to implement the distributed testbed.

REFERENCES

- [1] V. Cisco, "Cisco visual networking index: forecast and methodology 2016–2021." <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf>, 2017. [Online; accessed 20-March-2019].
- [2] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Presented as part of the 2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, vol. 12, pp. 1–6, 2012.
- [3] Z. Cai, A. L. Cox, and T. Ng, "Maestro: a system for scalable openflow control." <http://hdl.handle.net/1911/96391>, 2010. [Online; accessed 10-January-2019].
- [4] D. Erickson, "The Beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 13–18, ACM, 2013.
- [5] N. Telegraph, "Telephone corporation, Ryu network operating system," 2012.
- [6] P. Floodlight, "Floodlight openflow controller." <http://www.projectfloodlight.org/floodlight/>. [Online; accessed 20-March-2019].
- [7] T. Koponen *et al.*, "Onix: a distributed control platform for large-scale production networks," in *OSDI*, vol. 10, pp. 1–6, 2010.
- [8] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: wait-free coordination for internet-scale systems," in *USENIX annual technical conference*, vol. 8, Boston, MA, USA, 2010.
- [9] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pp. 3–3, 2010.
- [10] J. Stribling *et al.*, "Flexible, wide-area storage for distributed systems with WheelFS," in *NSDI*, vol. 9, pp. 43–58, 2009.
- [11] F. Benamrane, M. Ben mamoun, and R. Benaini, "An east-west interface for distributed SDN control plane: implementation and evaluation," *Computers & Electrical Engineering*, vol. 57, pp. 162–175, 2017.
- [12] Y. Fu *et al.*, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 117–131, 2015.
- [13] K. Phemius, M. Bouet, and J. Leguay, "Disco: distributed multi-domain SDN controllers," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–4, IEEE, 2014.
- [14] K. Phemius, M. Bouet, and J. Leguay, "Disco: distributed SDN controllers in a multi-domain environment," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–2, IEEE, 2014.
- [15] P. Berde *et al.*, "Onos: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, ACM, 2014.
- [16] "Titan distributed graph database." <https://github.com/thinkaurelius/titan>, Oct. 2018. [Online; accessed 20-March-2019].
- [17] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.
- [18] P. OpenDaylight, "OpenDaylight controller." <https://www.opendaylight.org/>. [Online; accessed 20-March-2019].
- [19] "Project proposals: ODL-SDNi App - OpenDaylight project." https://wiki.opendaylight.org/view/Project_Proposals:ODL-SDNi_App. [Online; accessed 20-March-2019].

- [20] “ODL-SDNi developer guide — OpenDaylight documentation Oxygen documentation.” <https://docs.opendaylight.org/en/stable-oxygen/developer-guide/odl-sdni-developer-guide.html>. [Online; accessed 20-March-2019].
- [21] Object Management Group (OMG), “Data distribution service (DDS).” <http://www.omg.org/spec/DDS/1.4>, Apr. 2015. [Online; accessed 10-January-2019].
- [22] L. Bertaux, A. Hakiri, S. Medjiah, P. Berthou, and S. Abdellatif, “A DDS/SDN based communication system for efficient support of dynamic distributed real-time applications,” in *2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, pp. 77–84, IEEE, 2014.
- [23] A. Llorens-Carrodegua, C. Cervello-Pastor, I. Leyva-Pupo, J. M. Lopez-Soler, J. Navarro-Ortiz, and J. A. Exposito-Arenas, “An architecture for the 5G control plane based on SDN and data distribution service,” in *2018 Fifth International Conference on Software Defined Systems (SDS)*, pp. 105–111, IEEE, 2018.
- [24] “RedIRIS.” <https://www.rediris.es/lared/>. [Online; accessed 20-March-2019].
- [25] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, “The internet topology zoo,” *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.